CSC 2401 Introduction to Computer Complexity - Fall 2001
Lecture #3, Thursday, September 20, 2001
Lecturer: Allan Borodin
Scribe: Francisco J. Estrada

Lecture #3

3.1 Review

Here's a brief review of the topics recently covered in lecture:

- We have discussed what it means to say that a Turing Machine m recognizes language L=$\mathcal{L}$(m) in time T(n) (and respectively, in space S(n)).
- We defined two very important complexity classes: Deterministic Time DTIME(T(n)); that is, the set of languages L for which there exists a Turing Machine $\mathcal{M}$ that recognizes L in time T(n).
- And Deterministic, DSPACE(S(n)); defined in a similar way
- Two additional special complexity classes were defined:
- **DL**=**DLOG**=$\cup_{c\geq 0} DSPACE(c \cdot \log n)$=$DSPACE(\log n)$.
- **P**=POLYTIME=$\cup_{k\geq 0} DTIME(n^k)$.

3.2 Tape Compression

For the first of the special complexity classes, **DLOG**, it can actually be proven that $DSPACE(c \cdot \log n) = DSPACE(\log n)$ for any $c > 0$. The important point is to realize that the space required by any Turing Machine $\mathcal{M}$ with a set of work tape symbols Γ, can be "compressed" by using a larger set of work tape symbols Γ'; hence:

$\forall\ c > 0$ if $\mathcal{M}$ recognizes L in space S(n), then $\exists\ \mathcal{M}'$ which recognizes L in space $[c \cdot S(n)]$.

For example, consider a Turing Machine $\mathcal{M}$ whose tape is encoded in binary (its set of work tape symbols is Γ={0,1}). Now consider a Turing Machine $\mathcal{M}'$ whose tape is encoded using hexadecimal values (its set of work tape symbols is Γ'={0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F}). Whatever $\mathcal{M}$ encodes in its tape, can be encoded by $\mathcal{M}'$ using only one fourth as much space by replacing each group of four symbols in $\mathcal{M}$'s tape with the appropriate hex symbol in $\mathcal{M}'$'s tape (Figure 1).

3.3 Time speed-up

In a similar way:

$\forall\ c > 0$, if $\mathcal{M}$ recognizes L in time T(n), then $\exists\ \mathcal{M}'$ which recognizes L in time $[c \cdot T(n) + n]$.

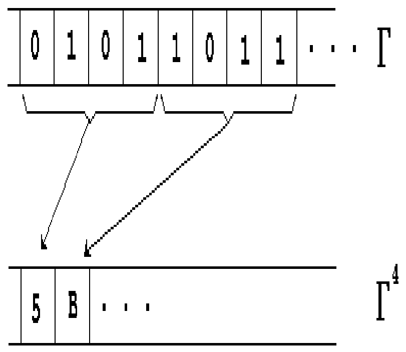

Figure 1: Tape Compression

This is called time speed-up, and the idea behind it is that the work done by a Turing Machine m can be simulated by a Turing Machine $\mathcal{M}'$ in such a way that several steps of m can be carried out in just one step by $\mathcal{M}'$.

3.4 Relation between a k-tape TM, and a 1 tape TM

Theorem:

Let $\mathcal{M}$ be a k-tape Turing Machine recognizing L=$\mathcal{L}$ ($\mathcal{M}$) in time T(n) (and respectively, in space S(n)); then there is a one-tape Turing Machine $\mathcal{M}'$ recognizing L=$\mathcal{L}$ ($\mathcal{M}$) in time $(T(n) \cdot S(n)) \leq (T(n))^2$ (respectively, in space S(n)).

The one-tape machine needs to encode the elements in all the tapes of the k-tape machine, this is similar to the tape compression procedure, except that now the elements are grouped *across* the k-tapes of the original machine (Figure 2). Additionally, it must encode the position of the heads of the original machine; hence it will have a set of work tape symbols $\Gamma = (\Gamma \cup \Gamma')^k$.

At each step, the one-tape machine has to "remember" what needs to be done on each of the tapes of the original k-tape machine, and perform each of the necessary operations. This may involve a displacement along the full length of its work tape, hence the time required by this machine to recognize L is now $(T(n) \cdot S(n))$.

3.5 Relation between a k-tape TM and a 2-tape TM

Theorem:

Let a k-tape machine $\mathcal{M}$ recognize L=$\mathcal{L}$ ($\mathcal{M}$) in time T(n), then there is a 2-tape machine $\mathcal{M}'$ recognizing L=$\mathcal{L}$ ($\mathcal{M}$) in time $T(n) \log T(n)$.

Aside:

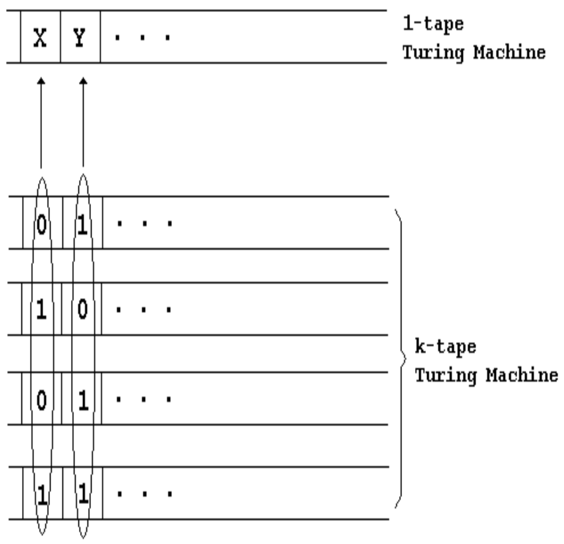

Figure 2: Compression across k-tapes

$\mathcal{M}'$ works obliviously in the sense that the position of the tape heads depends on $n = |x|$ but not on x.

3.6 Philosophical (Theological) aside

- *Church-Turing Thesis*

  A function computable in any reasonable model of computation, is computable by a Turing Machine.

- *Extended Church-Turing Complexity Thesis*

  Any problem computable in POLYTIME by a "classical" (used to say "reasonable") model of computation, is computable in POLYTIME by a Turing Machine (even a 1-tape TM).

  But, what do we mean by "classical"?

  1.- a RAM with +, -, $\leq$, etc. but no *.

  or

  2.- a RAM with * if we charge for basic operations, for example, charging ℓ steps for performing the operation when the maximum length of any argument has length ℓ.

  This extra cost is necessary because with * it becomes possible to compute very large numbers in a small number of steps. However, these large numbers must be encoded and stored in any classical machine, and if the number has length ℓ, it must take at least ℓ steps to perform any operation on it.

For example, with *, it becomes possible to compute 2^{2^t} in only t operations, however, this number requires 2^t bits to encode, and hence, performing any operation on it must take at least 2^t steps.

3.7 Expanded Complexity Thesis

The previous definitions of complexity classes allow us to study which functions can be efficiently computed. In general we equate "efficiently computable" with POLYTIME. However, this definition is suspect; consider the following case:

Consider two functions, one with complexity n^{10^6} (in POLYTIME) and one with complexity $(1.0000000......1)^n$ (exponential) where the decimal point is followed by 10^6 zeros. It is evident that for any reasonable value of n, the function with exponential complexity is better that the one in POLYTIME. It must be noted that such cases (i.e. of these functions being complexity bounds of useful algorithms) are rare (or maybe non-existant), and hence the former definition of "efficiently computable" is accepted.